Project 2: Classification
CSE 574

Siddhant Rane
UB ID: 5009 7974

December 6, 2013

1 Objective

This project is to implement and evaluate classification algorithms. We implement two different methods of classification and compare their performance along with a publicly available package. The classification models used are :

(a) Logistic Regression
(b) Neural Networks
(c) Neural Network Package

We perform classification of Handwritten Numerals. The project involves Multi-class Classification where Each DIGIT is a class. The input data was provided in two formats: GSC features Extracted from each image where each image is represented with 512 different features. The other format is Handwritten digit images which can be used with any other feature extractors.

MULTI CLASS LOGISTIC REGRESSION:

$$\mathrm{p}(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}.$$

where the 'activations' a_k are given by $a_k = w_k^T \phi$

NEURAL NETWORK:

$$y_k(x, w) = f\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)})\right)$$

where in our case $h(.)$ is either sigmoid or tanh activation function and $f(.)$ is used as soft-max function.

2 How to Run:

2.1 LOGISTIC REGRESSION

Step 0: Perform clear all, close all operations.
Step 1: load the **project2_data.mat** file. It is a .mat file containing required datasets.
Step 2: To train the model type: **w= train_lr(phi)**
Step 3: After the training, you will receive the **w** for training set as output. This means the training is done.
Step 4: To use the predict function, type: **labels=test_lr(w,phi_test)**
Step 5: The predict function will output class labels for test set and print the values of Error Rate and Reciprocal Rank.

```
>> train_lr(phi);

error =

    3.0065

>> test_lr(w,phi_test);
the Error Rate for the Multi Class Logistic Regression model is 2.466667e+00
the Reciprocal Rank for the Multi Class Logistic Regression model is 1
>>
```

Figure 1: Running Logistic Regression

2.2 NEURAL NETWORK

Step 0: Perform clear all, close all operations.
Step 1: load the **project2_data.mat** file. It is a .mat file containing required datasets.
Step 2: To train the model type: **[w1 w2]= train_nn(phi)**
Step 3: After the training, you will receive the **w1 and w2** for training set as output. This means the training is done.
Step 4: To use the predict function, type: **labels=test_lr(w1,w2,phi_test)**
Step 5: The predict function will output class labels for test set and print the values of Error Rate and Reciprocal Rank.

3 Logistic Regression

In our case we implement Multi Class Logistic Regression. Hence we use the softmax function defined earlier. $\mathrm{p}(C_k|\phi) = y_k\phi = \dfrac{\exp(a_k)}{\sum\limits_j \exp(a_j)}$ where the 'activations' a_k are given by $a_k = w_k^T\phi$. We **do not** use the sigmoid function $\sigma(1-\sigma)$ as it is useful for Two Class classification generally.

Hence we need to train the Model to get appropriate weights w_k.

In order to train the model we need to minimize the Error on the training dataset. To calculate the error we use a new function called the **cross-entropy function** which is used for multiclass classification. It is given as:

$$E(w_1 \dots w_k) = -\ln p(T|w_1 \dots w_k) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

In order to minimize the Error, we use an approach of gradient descent. The formula for gradient descent is given as :

$$w^{\tau+1} = w^{\tau} - \eta \bigtriangledown E_n$$

which basically means $w_{new} = w_{old} - \eta \bigtriangleup E$.
Here τ is time which could be considered as iteration in our case. Hence we reduce the error sequentially by updating weights and calculating error in each iteration.

3.1 Parameters

W: W is the weight matrix. The dimensions of the weight Matrix are $D + 1 \times K$. Here D is the number of input features. In our case the GSC extracts 512 bits as features. Hence $D = 512$. Each Feature has a value either 0 or 1. We use D+1 because we add a bias to the input matrix ϕ where all values w_0 are set to 1. K is the number of classes which in our case is 10. Since there are 10 digits from 0 to 9, we have one class for each digit. We can initialize the weight matrix by either taking all elements as zero and then train them or take random values between -1 and 1. The random values can be generated using rand() function in MATLAB. In our case, we have Initialized all weights to 0. Both approaches are valid and it does not affect the final weights in any way.

Step Size η: This can be considered one of the Most Important Parameters to decide for Algorithms that use Gradient Descent. η is used in the formula of updating weight values and hence it determines how much change should be made in weights in each iteration.

A LARGE step size η means the weight values will change abruptly and may skip over minima. Hence the Error value will change with large variation. Hence it could decrease fast but could also increase suddenly or miss a minimum.

A SMALL step size η means the Weights are changed gradually and the error will change gradually as well. However the drawback is that the algorithm will converge slowly and after a lot more iterations. Hence amount of time required to reach a minima will be large. The minima found is not guaranteed to be a global minima in either cases. With small step size it is possible to reach the exact minimum value without skipping over the minimum.

VARIABLE Step Size η is a good approach to take. However it does not mean we gain only the advantages of both approaches. It is possible in the worst case to suffer from drawbacks of both the above approaches. Hence it is Important to vary the step size in an intelligent and efficient way. We Choose to take this approach and vary the Step Size as and when needed.

Variable Step Size Implementation:

```
if old_error<=error
    stepper=0;
    step=0.0001;

    else
    stepper=stepper+1;
    if stepper>5
        step=step+0.0005;
        stepper=0;
    end


    w_star=w_new;
    old_error=error;

end
```

Step size is initialized to a very small value. If error drops consecutively for certain number of steps at current step size we increase the step size. If at any point error does not drop, we quickly reduce step stize to initial value. This algorithm definitely converges faster than fixed step size which we shall see later.

Error Gradient $\triangle E$: It is given by the formula :

$$\bigtriangledown_{w_j} E(w1, \dots, w_k) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\phi_n$$

We calculate the Error gradient after each iteration which is basically the difference between true output values and our predicted values of the training set. The Error is then used to change the values of the weight matrix W. The step size allows us to control the change in the Weight matrix since values of $\triangle E$ can vary significantly.

Gradient Count : This basically is a counter which we have used to Determine the **number of iterations**. In order for the algorithm to converge at some point we must select a Maximum Number of Iterations or the Algorithm will continue to run for a very long time. Hence to limit the number of iterations we maintain Gradient Count variable. The value is incremented at each iteration and checked before each iteration to see if it has exceeded the maximum number of iterations allowed. A large Maximum value means longer execution time but better accuracy.

3.2 Logistic Regression: Experimental Phase

Training Set	#19978 Samples
Test Set	#1500 Samples

3.2.1 Phase 1: Training

Function: **train_lr(phi)**
phi (ϕ): $N \times (D+1)$ Matrix of Feature Vectors where
N: number of Samples i.e. 19978 and
D: Number of features i.e. 512.

In the training phase, we use the following formulae to find Weight Matrix W^* :

$$a_k = w_k^T \phi$$

where a_k is the 'activation' function. W is initialized to zero and phi is our input feature vector.

$$\mathrm{p}(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

This is the Soft-Max function used for Multi Class Classification where $y_k(\phi)$ is the Posterior Probability. It is the output predicted by our model based on the weight matrix.

$$E(w_1 \dots w_k) = -\ln p(T|w_1 \dots w_k) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

This is the cross-Entropy error function used to calculate the error. We continue Gradient descent until error reaches an acceptable value or number of iterations exceeds maximum.

$$\bigtriangledown_{w_j} E(w1, \dots, w_k) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\phi_n$$

$\bigtriangledown_{w_j} E(w1, \dots, w_k)$ is the error gradient also known as loss function. t_{nj} are the Ground Truth values of training set.

$$w_{new} = w_{old} - \eta \bigtriangleup E$$

The $\bigtriangledown_{w_j} E(w1, \dots, w_k)$ we calculated earlier is now used to update the values of the Weight matrix. η is the step size which we vary in order to control the change in the weight matrix in each iteration. This formula allows sequential updating of weights to minimize error and hence is called Gradient Descent.

ALGORITHM:

Step 1: Initialize weight matrix of 513 $\times$ 10 with all elements zero and step size to 0.0001.

Step 2: Calculate activation function a_k.

Step 3: Calculate Posterior Probability using softmax function.

Step 4: Calcualte cross-entropy Error for predicted posterior probability.

Step 5: Calculate Error gradient also called loss function.

Step 6: Update Weight Matrix if error is not acceptable and increment gradient count.

Step 7: Repeat Steps 2 to 6 until Error is not acceptable or number of iterations exceeds limit.

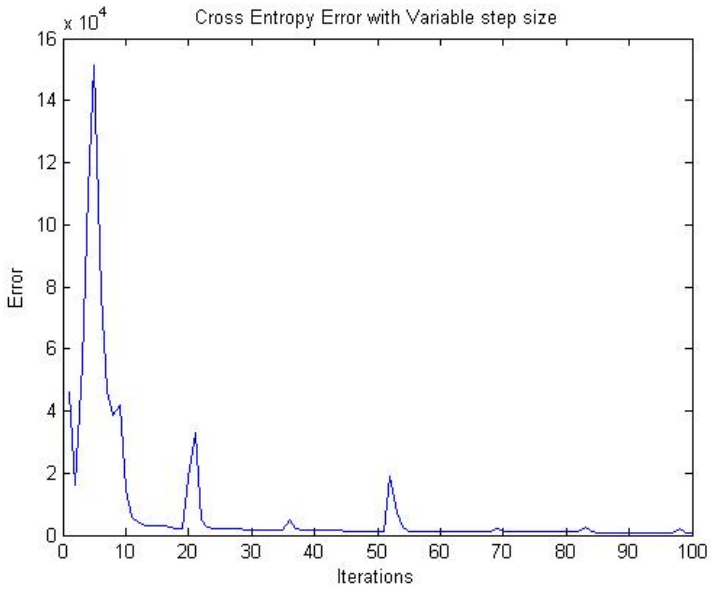

Figure 2: Iterations 1-100

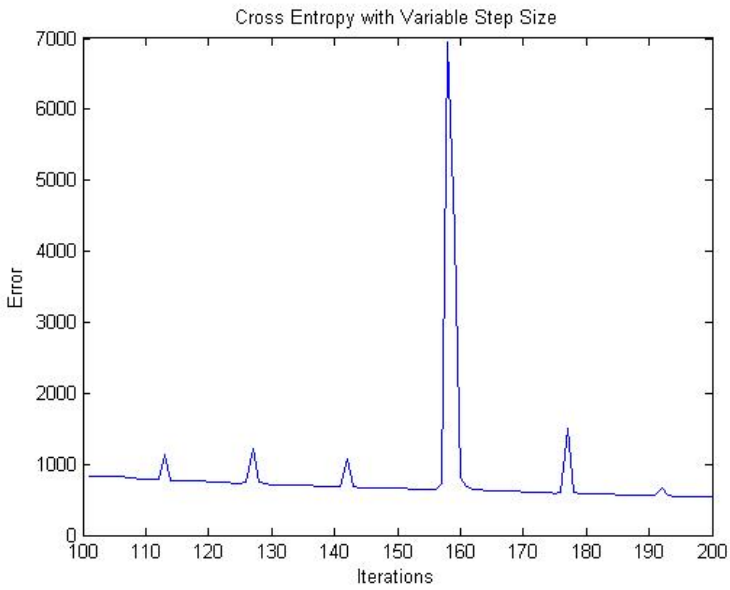

Figure 3: Iterations 101-200

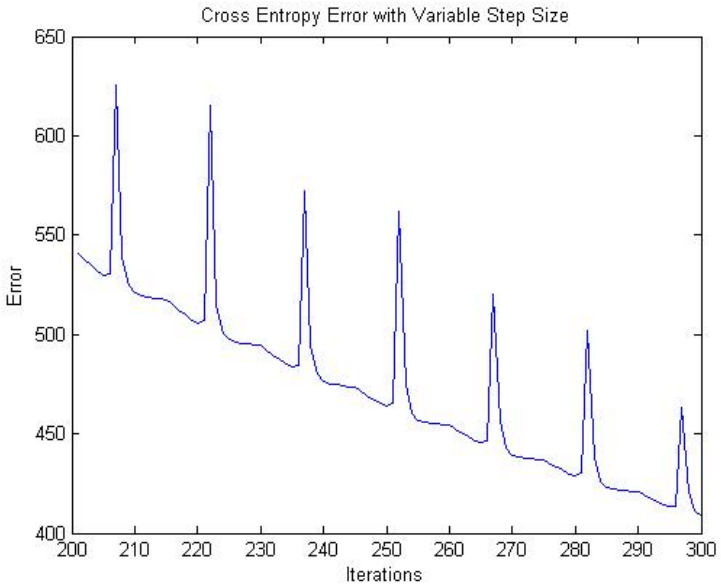

Figure 4: Iterations 201-300

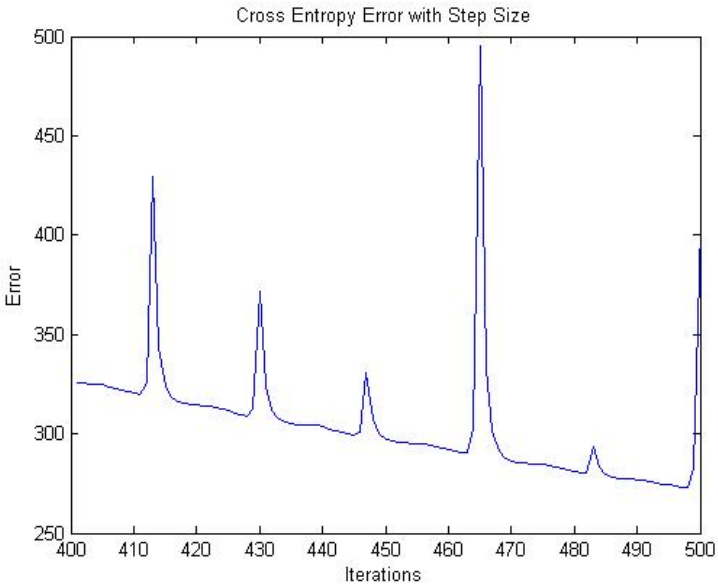

Figure 5: Iterations 301-400

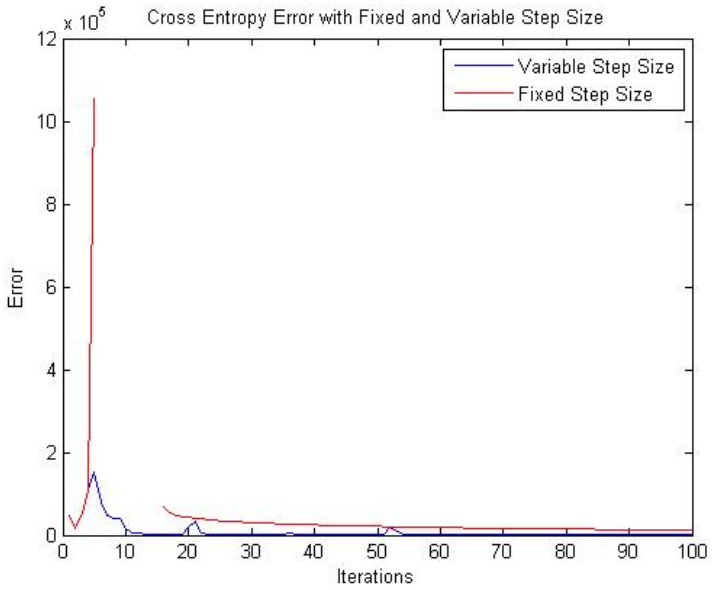

Figure 6: Iterations 1-100

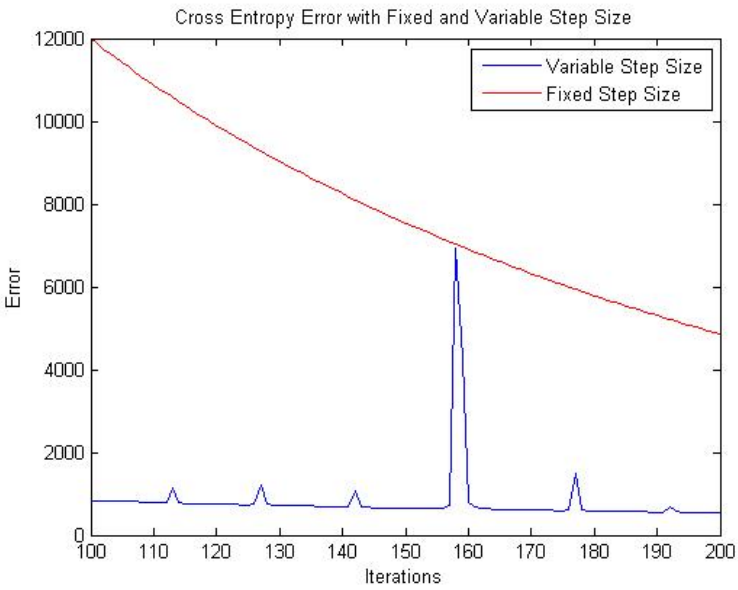

Figure 7: Iterations 100-200

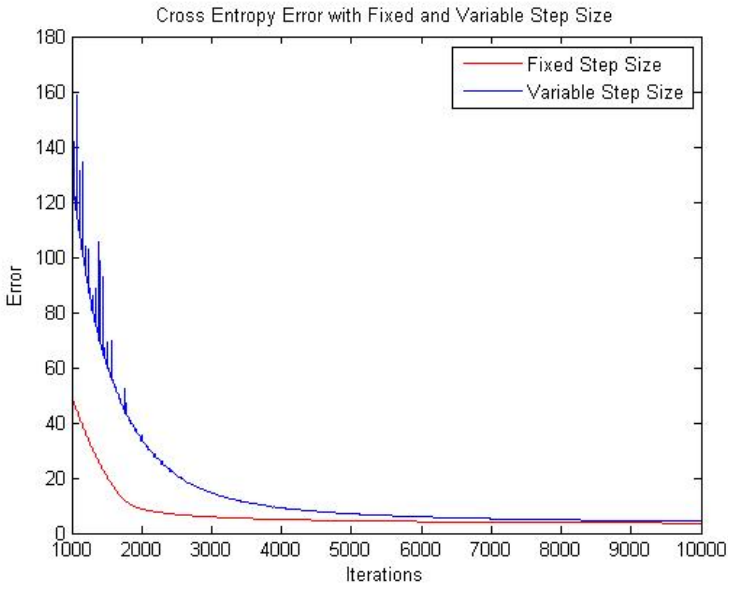

Figure 8: Iterations 1000-10000

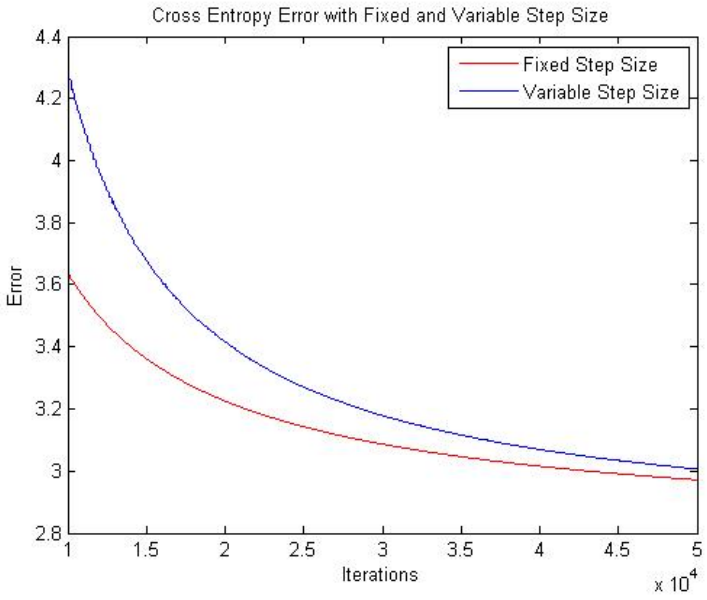

Figure 9: Iterations 10000-50000

3.2.2 Phase 2: Testing Phase

Function: test_lr(W,phi_test)

W^*: Is the final Weight Matrix that we have calculated after training. It is a $(D+1) \times K$ matrix

phi_test(ϕ_t): Is a $N\times$ (D+1) Matrix of Feature Vectors of Test Set for which we will predict Class.

N: number of Samples i.e. 1500 and
D: Number of features i.e. 512 which obviously remains the same
K: Number of ouput Classes i.e. 10

In the testing phase or prediction phase, we use the basic formula:

$$y = w^* \times \phi$$

The formula becomes output= $\phi \times w^*$ in our case due to dimensions.

ALGORITHM:

Step 1: Calculate output= $\phi \times w^*$.

Step 2: Calculate the largest value in each Row and the Column that it belongs to using ind2sub function in MATLAB.

Step 3: Initialize a Prediction Matrix of dimensions N×K with all zeros.

Step 4: For each row of Prediction Matrix, change element of the column that had largest value in Output Matrix to 1.

Step 5: This Prediction Matrix is now compared to the ground Truth values of Test Set.

Step 6: Calculate Error Rate which is the percentage of Miss Classified values.

Step 7: Calculate Reciprocal Rank which is the multiplicative inverse of rank of the first correct classification label.

OUTPUT:

the Error Rate for the Multi Class Logistic Regression model is 2.466667e+00.
This means 2.46% of the 1500 values were misclassified.
We consider this error acceptable considering 50,000 iterations only.

the Reciprocal Rank for the Multi Class Logistic Regression model is 1.
This means the very first predicted value was Correct which is the best possible outcome.

4 Neural Network

Neural Network function after combining the various stages can be written as follows:

$$y_k(x,w) = f\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)})\right)$$

We implement Neural Network with 1 Hidden Layer.

For the first Layer i.e. Input layer(h), we use the sigmoid function: $\sigma(1-\sigma)$
We could also use the tanh function.

For the hidden layer(f), we use softmax function because we have a Multi Class output.
We use the following formulae to train the Neural Network:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j)$$

$$\text{i.e. } z_j = \sigma(a_j)$$

The quantities a_j are called activations. Here $h(.)$ is the sigmoid function we use on the activations to get values of the hidden units z_j. The number of hidden units is a Parameter that we decide. Based on the value of M i.e. number of hidden Nodes and D, we have a $D+1 \times M$ weight Matrix $w_{ji}^{(1)}$.

Now we shall see the transformation corresponding to the second layer.

$$a_k = \sum_{i=1}^{D} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

where $k = 1 \ldots, K$ and K is the number of outputs, i.e. 10 in our case.

$$\mathrm{p}(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

This is the Soft-Max function used for Multi Class Classification where $y_k(\phi)$ is the Posterior Probability and where $j = 1 \ldots, K$ and K is the number of outputs. It is the output predicted by our model based on the weight matrix.

$$E(w_1 \ldots w_k) = -\ln p(T|w_1 \ldots w_k) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

This is the cross-Entropy error function used to calculate the error. We continue Gradient descent until error reaches an acceptable value or number of iterations exceeds maximum.

In Neural Networks, in order to perform gradient descent to minimize error, we perform the Back Propagation Algorithm. Backpropagation is more efficient for finding minima but even this algorithm does not guarantee a global minima. In order to perform backpropagation we must calculate the following values:

$$\delta_k = y_k - t_k$$

then we backpropagate to calculate deltas for hidden units.

$$\delta_j = z_j(1 - z_j) \sum_{k=1}^{K} w_{kj}\delta_k$$

We then calcualte the derivatives w.r.t. the first layer and second layer weights as follows:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

To minimize the error and optimize the weights, we use the gradient descent formula with the derivatives calculated above to calculate new weights w_{ji} and w_{jk}.

$$w^{(\tau+1)} = w^{(\tau)} - \eta \bigtriangledown E(w^{(\tau)})$$

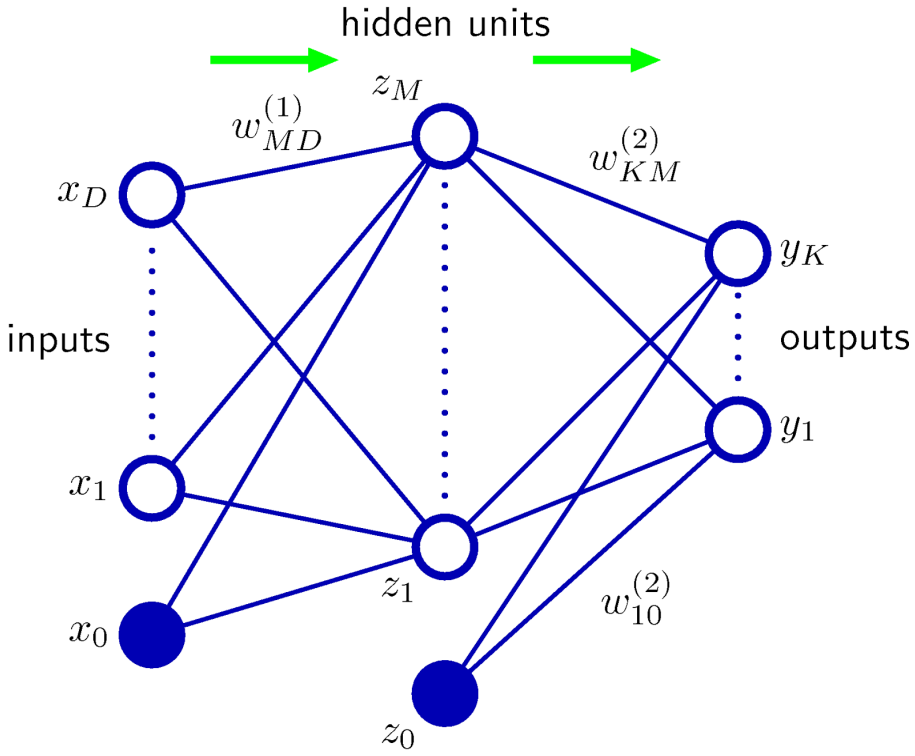

Figure 10: Neural Network

The above diagram depicts our Neural Network model.
The Input Layer has 512 Dimensions in our case. Hence D=512.
The Hidden Layer can have any number of nodes and is a Parameter that we should decide. It if generally acceptable to have,

$$\text{Hidden Nodes} = \frac{2}{3} \times \text{ Number of dimensions.}$$

Larger the M, greater the accuracy but at the cost of more complexity, computation and time to train.
In our case, there are 10 output classes corresponding to each Digit $0 - 9$. Hence the value of K in the above diagram will be 10.

4.1 Parameters

Number of Hidden Layers: It is believed that majority of applications do not require more than 2 Hidden layers. Infact a single hidden layer can be used to perform most problems. Our case is also one of these where a Single Hidden Layer is enough to perform Classification efficiently and accurately.
NO Hidden Layer: This can represent only linearly separable data.
ONE Hidden Layer: Approximate function from one finite space to another.
TWO Hidden Layers: Can represent arbitrary decision boundary. This model can represent functions with any kind of shape but requirement for such models is rare and complexity is higher.
Hence we choose ONE Hidden Layer model.

Number of Hidden Units M: The next parameter we must choose is M, i.e. the number of hidden nodes/units. Choosing the number of hidden units is important as they impact the final output.
Using a very small number of Hidden nodes can cause Underfitting and hence training error may not decrease to desired values. This also means, testing error will be high.

On the other hand, having a very large number of Hidden nodes can cause Overfitting where the model is tuned to training set and Error on testing set will be high. Having a large number of Hidden nodes also means higher complexity and higher training time. Hence the program will take significantly longer to train the model.
It is generally believed that,
$M = \frac{2}{3} \times$ Number of Features, is a good value for Hidden Units.
There are two other thumb rules for choosing M as follows,
The number of hidden nodes must be between number of input features and output classes
The number of hidden nodes must be less than twice the size of input layer.

Step Size η: We already discussed the importance and impact of choosing a step size in the Logistic Regression Section. The same applies here but in addition, since we have two layers in the Neural Network model, we also have two weights. This means, we have two Step Sizes for updating the two weight matrices. Hence we must choose the step sizes according to the weights and their corresponding Error Gradients. This has certain drawbacks discussed below.
A low step size means low learning rate, and hence the network learns very slowly.
A high step size means a very high learning rate, which causes values of the weights to change drastically, this does not lead to efficient learning of weights.
If we keep a fixed step size, then it required a large amount of trial and error to find a perfect or optimum step size.
Changing the step size based on the Error gradient is a commonly used approach but it has certain drawbacks. In different cases, we may need large or small step size for small gradient and similarly for large gradient, a small or sometimes large step size may be required.
We need a small step size for small gradient near local minima. However, when we initialize weights to small random values, we may need a large step size for small gradient.

4.2 Neural Network: Experimental Phase

Training Set	#19978 Samples
Test Set	#1500 Samples

4.2.1 Phase 1: Training

Function: **train_nn(phi)**
phi (ϕ): $N \times (D+1)$ Matrix of Feature Vectors where
N: number of Samples i.e. 19978 and
D: Number of features i.e. 512.

ALGORITHM:

Step 1: Initialize Weight Matrices, w_{ji} and w_{kj} with random values between -1 and 1 and Initialize Step Sizes η_1, η_2 to appropriate values.

Step 2: Calculate activation units a_j.

Step 3: Calculate hidden node values z_j using sigmoid function of activation units.

Step 4: Calculate activation units a_k which are for second layer.

Step 5: Apply soft-max function on activation units a_j to get $y_k(\phi)$.

Step 6: Calculate cross-entropy Error.

Step 7: Calculate δ_k and δ_j.

Step 8: Calculate derivatives of Error with respect to weights to get $\frac{\partial E_n}{\partial w_{ji}^{(1)}}, \frac{\partial E_n}{\partial w_{kj}^{(2)}}$.

Step 9: Update the Weight values using gradient Descent Formulae.

Step 10: Continue Step 2 to Step 9 until we have an Error value equal to or below the desired value or until maximum Iterations are performed.

In the case of Neural Network, we use trial and error to find a fixed Step size where $\eta_1 = 10^{-7}$ and $\eta_2 = 10^{-4}$ are found to be GOOD values. However since it is a fixed Step Size the Algorithm takes longer to converge. Hence we take a variable Step Size after Initializing them with above values and using the following Code:

```
if old_error<=error
    stepper=0;
    step2=0.0000001;
    step1=0.0001;
    else
    stepper=stepper+1;
    if stepper>5
        step2=step2+0.0000005;
        step1=step1+0.0005;
        stepper=0;
    end
```

Here, we increment step size if error is decreasing 5 times consecutively. The amount to increment is different for both. We reset the step size if error does not decrease.

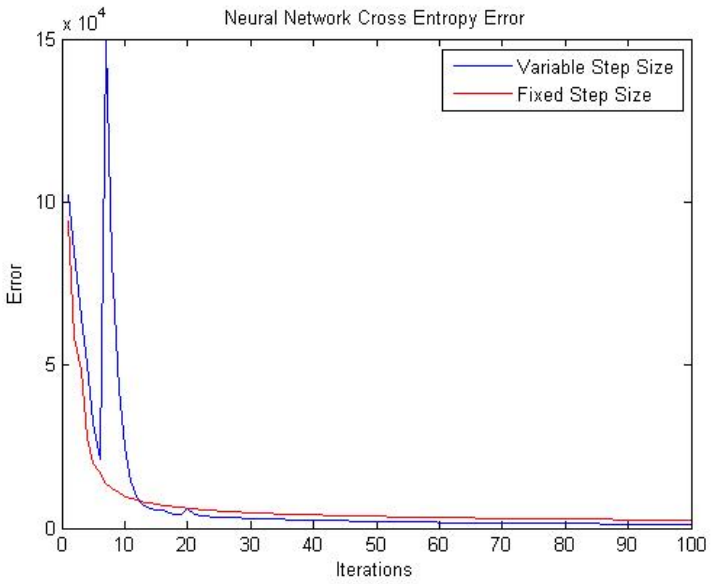

Figure 11: Iterations 1-100

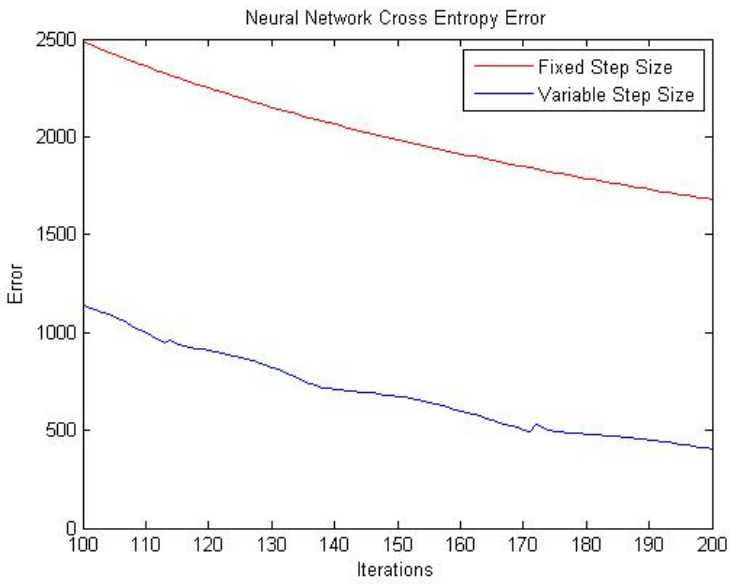

Figure 12: Iterations 100-200

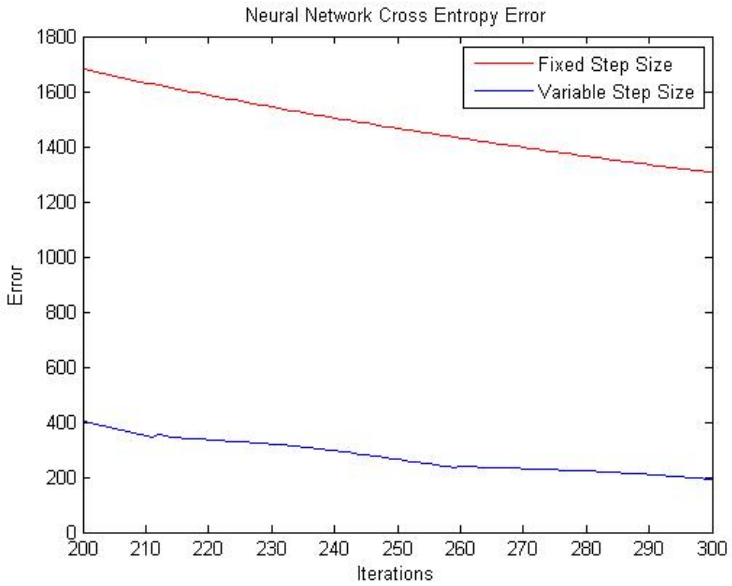

Figure 13: Iterations 200-300

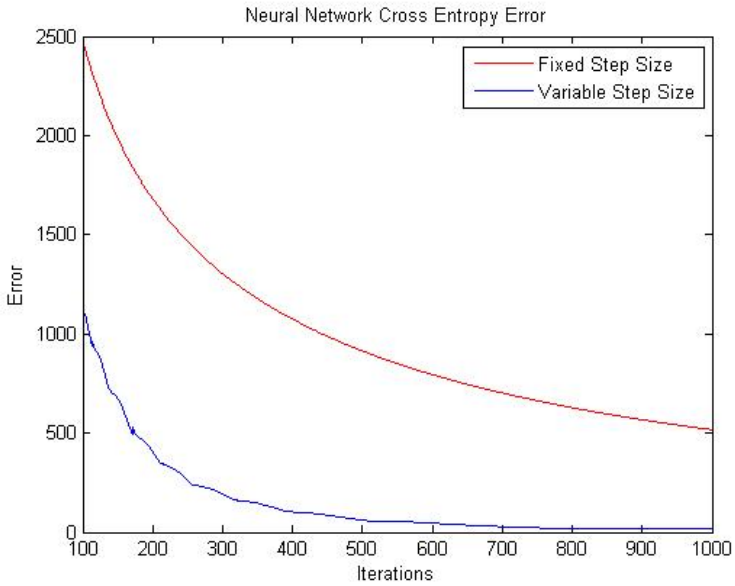

Figure 14: Iterations 100-1000

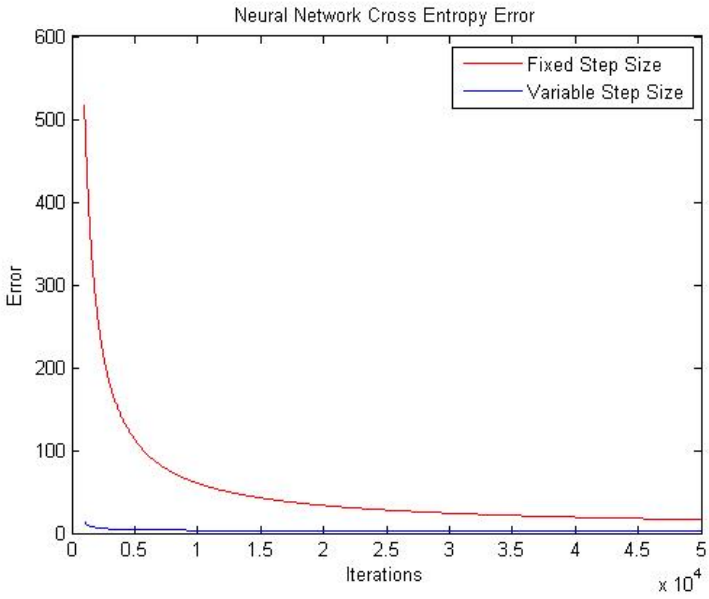

Figure 15: Iterations 1000-50000

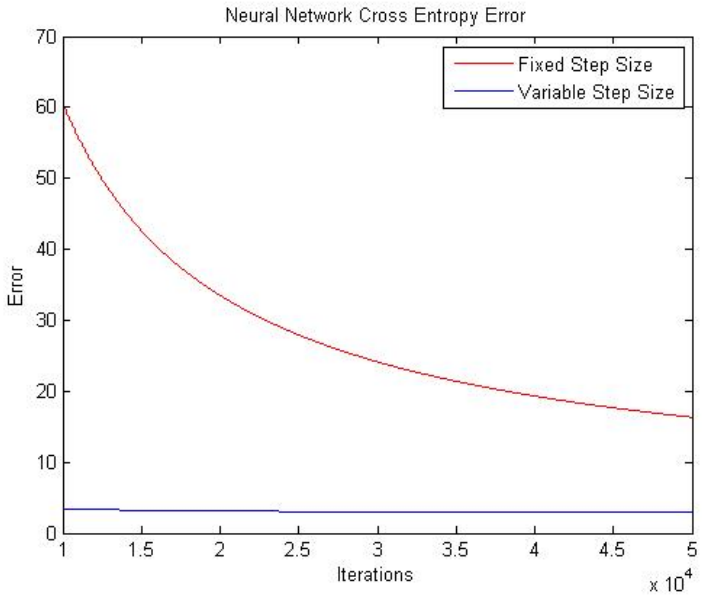

Figure 16: Iterations 10000-50000

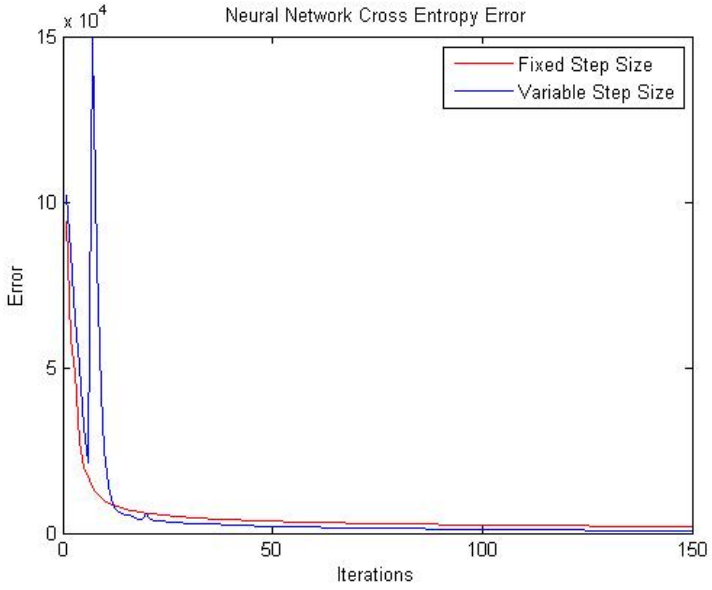

Figure 17: Iterations 1-150

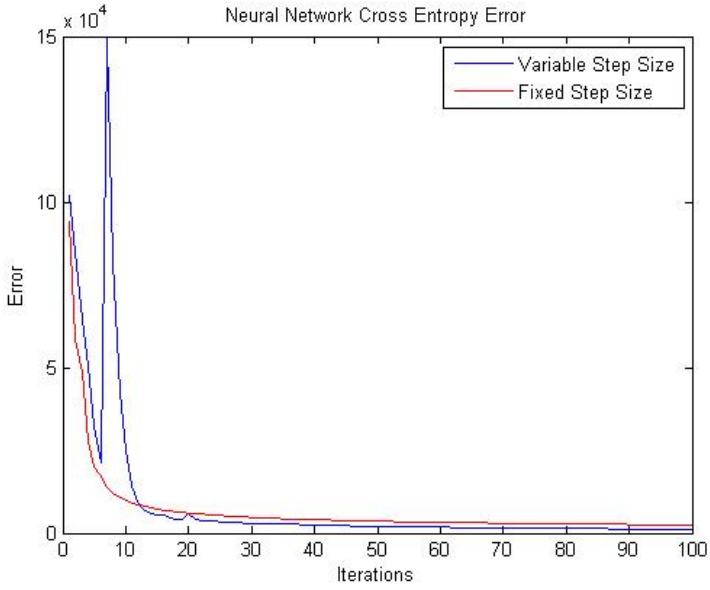

Figure 18: Iterations 1-100

4.2.2 Phase 2: Testing

Function: test_nn(w1,w2,phi_test)

$w1^*$: Is the first layer Weight Matrix that we have calculated after training. It is a $(D+1) \times M$ matrix

$w2^*$: Is the second layer Weight Matrix that we have calculated after training. It is a $M \times K$ matrix

phi_test(ϕ_t): Is a $N\times$ (D+1) Matrix of Feature Vectors of Test Set for which we will predict Class.

N: number of Samples i.e. 1500 and
D: Number of features i.e. 512 which obviously remains the same
M: Number of Hidden units in Hidden layer. i.e. 342 in our case
K: Number of ouput Classes i.e. 10

In the testing phase or prediction phase, we use the same formulae:

$$a_j = w^{(1)} \times \phi_{test}$$

$$z_j = \sigma(a_j)$$

$$a_k = w^{(2)} \times z_j$$

$$\mathrm{p}(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

This is the Soft-Max function used for Multi Class Classification where $y_k(\phi)$ is the Posterior Probability and where $j = 1 \ldots, K$ and K is the number of outputs. It is the output predicted by our model based on the weight matrices.

ALGORITHM:

Step 1: Calculate activation units of first layer i.e. a_j.

Step 2: Calculate hidden unit values i.e. z_j.

Step 3: Calculate activation units of second layer i.e. a_k.

Step 4: Calculate output predication values i.e. y_k.

Step 5: Calculate the largest value in each Row and the Column that it belongs to using ind2sub function in MATLAB.

Step 6: Initialize a Prediction Matrix of dimensions N×K with all zeros.

Step 7: For each row of Prediction Matrix, change element of the column that had largest value in y_k Matrix to 1.

Step 8: This Prediction Matrix is now compared to the ground Truth values of Test Set.

Step 9: Calculate Error Rate which is the percentage of Miss Classified values.

Step 10: Calculate Reciprocal Rank which is the multiplicative inverse of rank of the first correct classification label.

OUTPUT:

the Error Rate for the Neural Network model is 1.800000e+00.
This means 1.8% of the 1500 values were misclassified.
We consider this error acceptable considering 50,000 iterations only.

the Reciprocal Rank for the Multi Class Logistic Regression model is 1.
This means the very first predicted value was Correct which is the best possible outcome.

5 Neural Network Package

5.1 Why Choose Neural Networks

Neural Network Model is inspired from how a human brain works, where millions of neurons which have very small individual computing power communicate using energy signals and have switching times slower than modern day computers. Inspite of these limitations the brain performs tasks like Speech and Face Recognition, learning etc. much more efficiently than any of the algorithms we currently use. Thus it is important to try and create algorithms which use this property of the brain to be used in the fields where a human brain performs exceedingly well.

Artificial neural networks have always fascinated people in the field of Computer Science. However, in the days when neural networks were first used they did not perform very difficult tasks. Today we can perform regression, classification etc. using neural networks. We can increase the complexity of the model using more hidden nodes.

There are two types of Neural Networks called Feed Forward NN and the Recursive NN. The feed forward neural network is faster and less complex to train but has limitations in its use. The Recursive Neural Network on the other hand can perform a lot more complex tasks but is very complex to train due to the recursive nature. The recursion here means that as opposed to other Neural Networks, we can branch from one layer to a previous layer and train the weights again. This allows us to modify the weights more easily but requires complex training algorithms. There are various types of neurons in neural networks like the Binary Threshold, Linear, Sigmoidal Neural Networks.

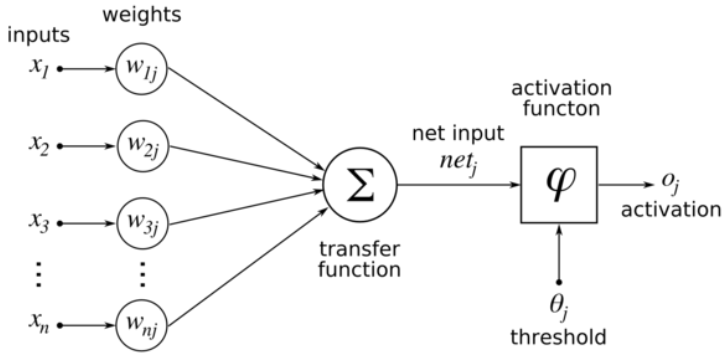

Figure 19: Neural Network

We now use the Neural Network Package available for regression using Classification tool on our data set now. We find the Error Rate for the Neural Network Package turns out to be 2.3 for the testing set after retraining it. Hence our implemented Neural Network has a better Error Rate of 1.8

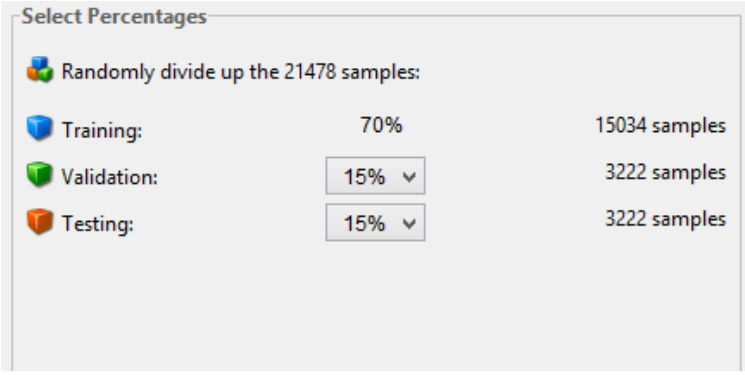

Figure 20: Neural Network Package Classification Tool

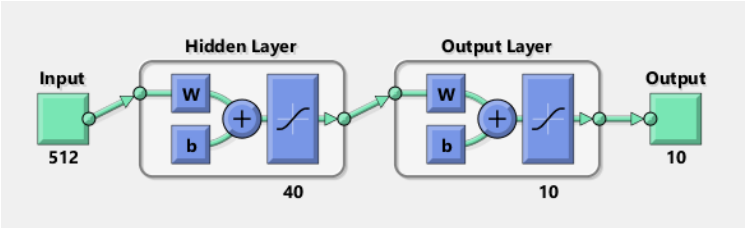

Figure 21: Neural Network Architechture

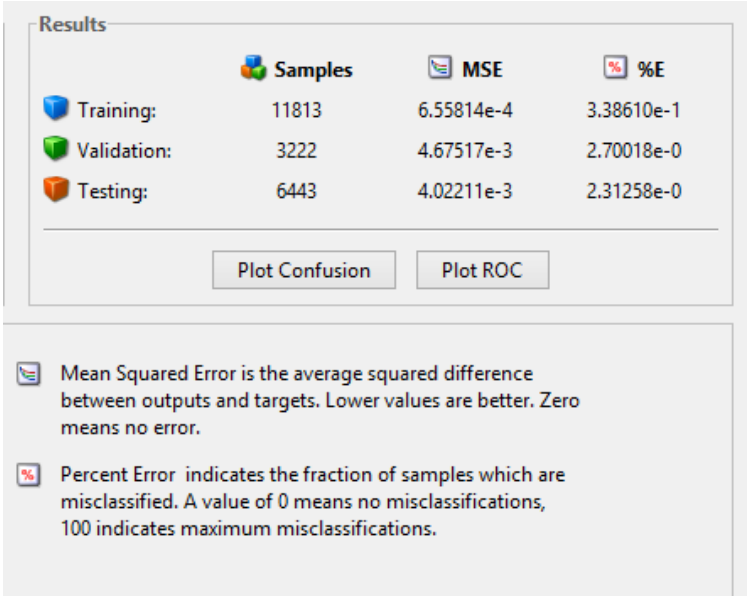

Figure 22: Errors Calculation

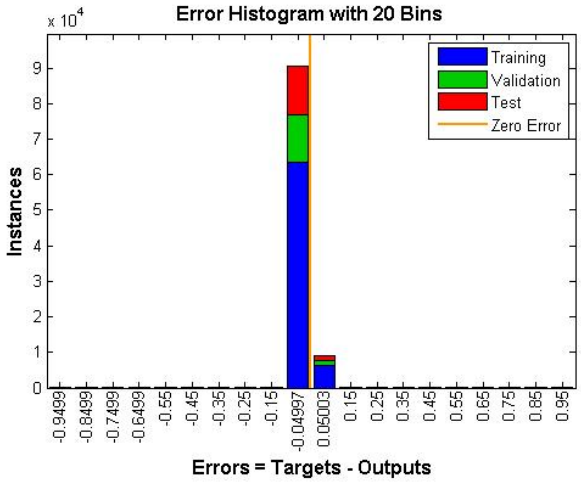

Figure 23: Error Histogram(values should be near zero error line)

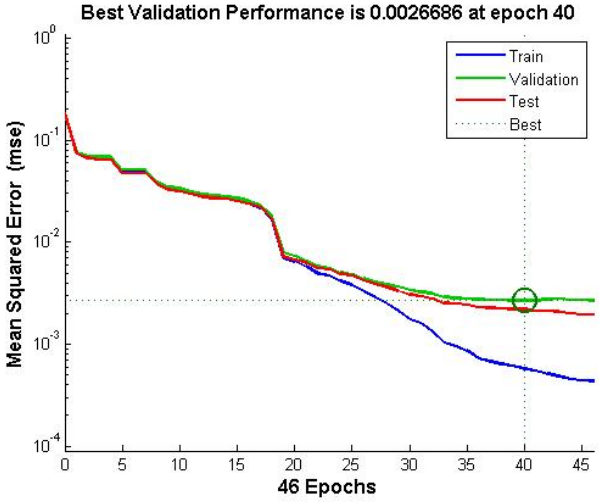

Figure 24: Error vs Epochs Peformance graph

6 Model Comparison

Model	Cross Entropy Error	Error Rate	Reciprocal Rank
Logistic Regression	3.0065	2.46 %	1
Neural Network	2.9216	1.8 %	1
Neural Network Package		2.31%	

We Observe from the above table and following graph that Neural network has a smaller Error, Error Rate and the Reciprocal Rank of both are same. Hence Neural Network performs better than Logistic Regression. The graphs also clearly show that Neural Network finds a minimum faster than Logistic Regression. Neural Network's Backpropagation algorithm is one of the reasons for faster performance. However, If speed of each Iteration is taken into account, Logistic Regression is faster to perform one iteration.

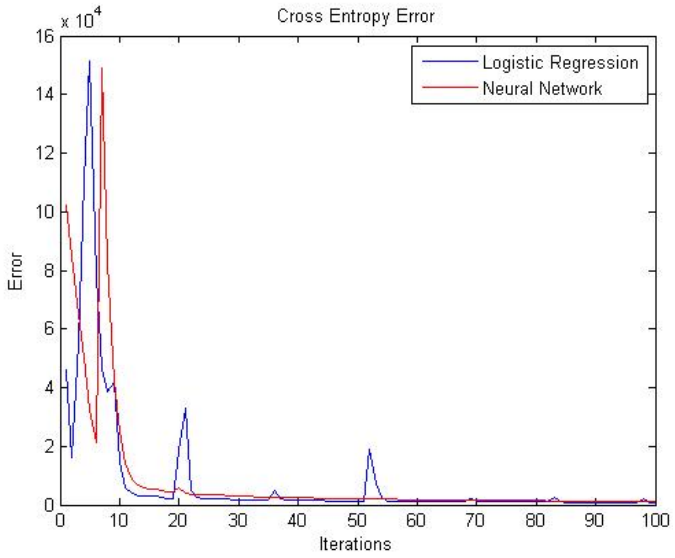

Figure 25: Iterations 1-100

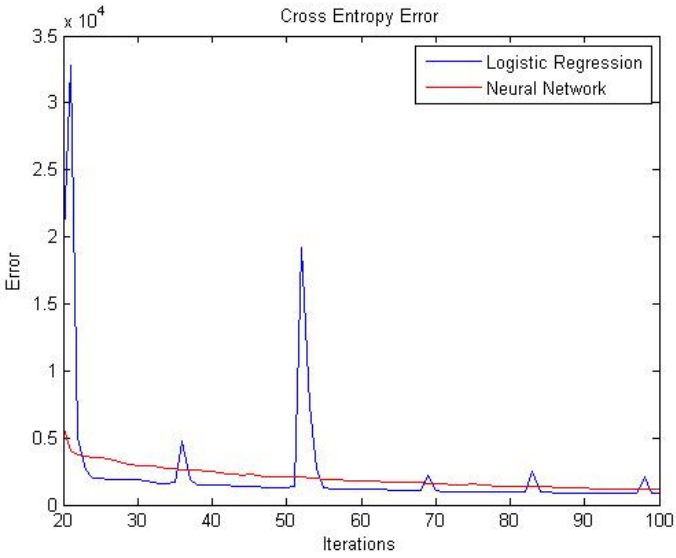

Figure 26: Iterations 20-100

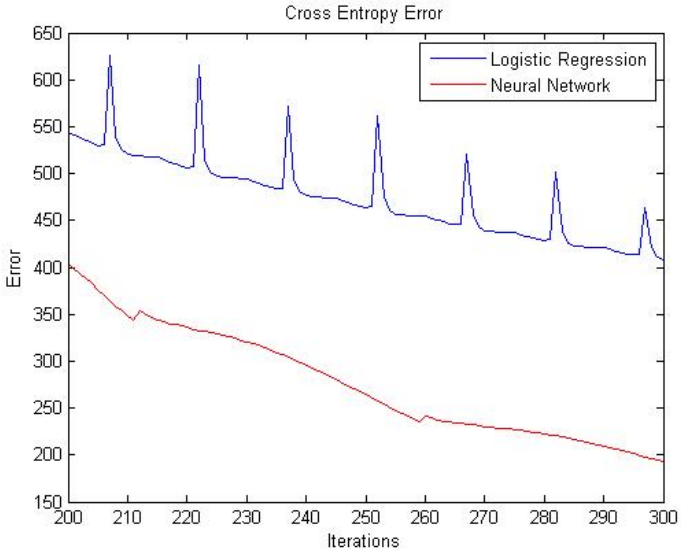

Figure 27: Iterations 200-300

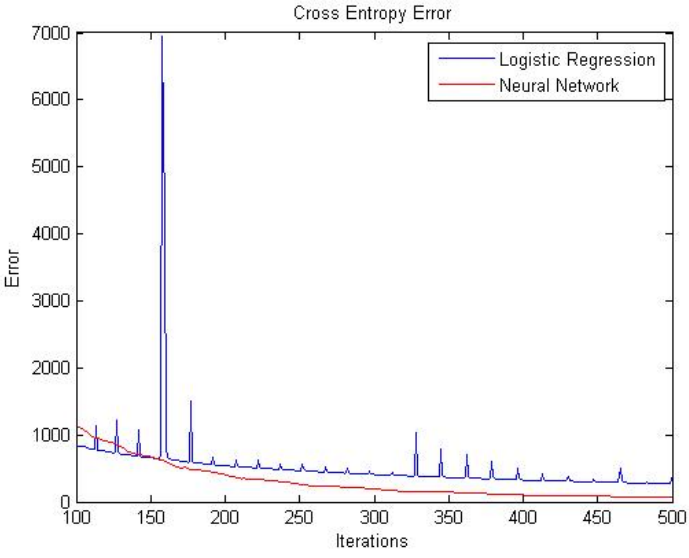

Figure 28: Iterations 100-500

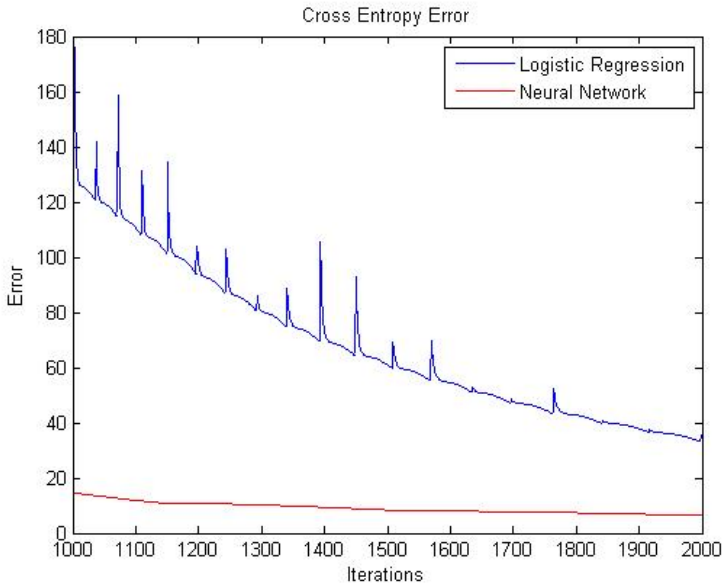

Figure 29: Iterations 1000-2000

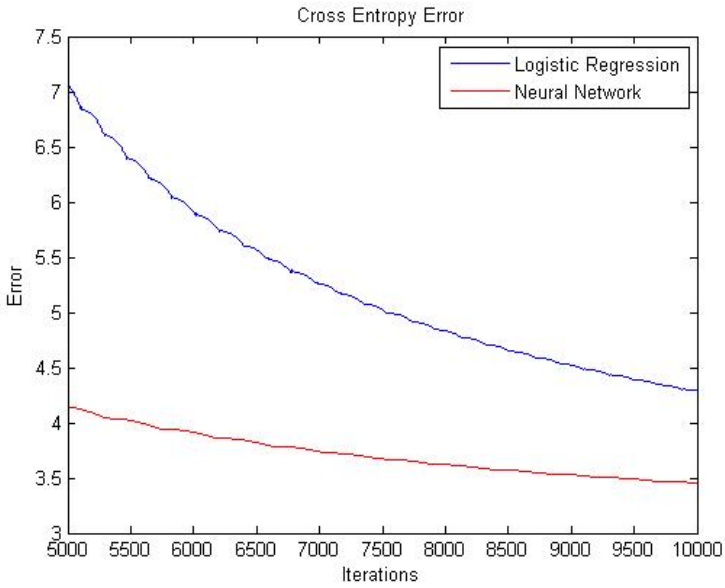

Figure 30: Iterations 5000-10000

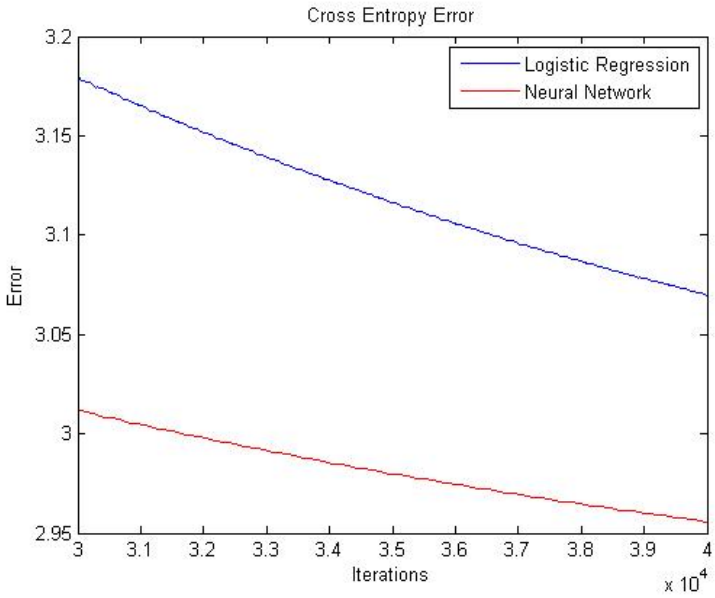

Figure 31: Iterations 30000-4000

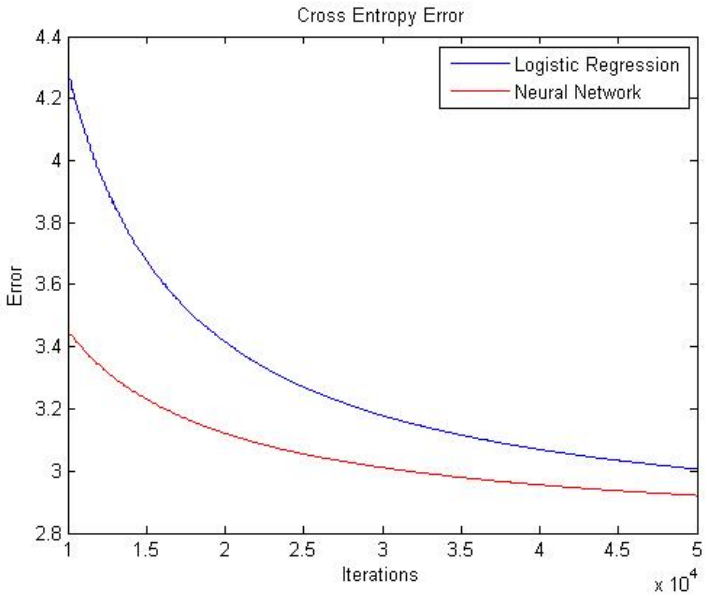

Figure 32: Iterations 10000-40000

6.1 Running statistics

Below are the Run time profile results of Neural Networks for Variable and Fixed Step Size respectively. It is clear that both take a lot of time to train using 50,000 Iterations.

Profile Summary

Generated 30-Nov-2013 15:45:47 using cpu time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
train_nn	1	50491.053 s	50491.031 s	
close	1	0.022 s	0.021 s	
close>safegetchildren	1	0.001 s	0.001 s	
close>getEmptyHandleList	1	0 s	0.000 s	
close>checkfigs	1	0 s	0.000 s	
close>request_close	1	0 s	0.000 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process

Figure 33: Neural Network Runtime Profile

Profile Summary

Generated 06-Dec-2013 13:52:55 using cpu time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
train_lr	1	20542.865 s	20542.865 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes
from the process of profiling.

Figure 34: Logistic Regression Runtime Profile

7 References

[1] Pattern Recognition and Machine Learning, CHRISTOPHER M. BISHOP.
[2] Introduction to Neural Networks for Java, Second Edition, JEFF HEATON.